By Dana Scott Kaufman

Working in Streams (without Getting Wet)

Using Streams to Read and Write Memo Field Data

Memo fields are common in applications that need to store an indefinite amount of character-based information in individual records in a database. Although memo fields are widely used, Delphi contains no easy way to import and export data from these fields. The DBMemo component is the only ready-made control that can access a memo.

But what if you need to store and manage large amounts of text, but don't include a *TDBMemo* object on any of your Delphi forms? For example, I regularly store program configuration data in string lists for easy manipulation at run time. The program must be able to load or store this information at any time.

Borland provides the means to perform these tasks, but they are poorly documented. This article will explain how to use streams to import and export data from fields in a database. Later, we'll address using these techniques with data stored on a SQL server, and the benefits of using these methods. Code examples are included that show how to read and write a string list to a memo field.

Into the Stream

To accomplish these data import/export feats, we must first become familiar with the concept of *streams*. Streams allow applications to read and write data sequentially to and from a medium that can store binary data. The stream can store the data in memory, on disk, or on other devices.

The main strength of streams is that the actual medium used is irrelevant. All streams work the same, in that the same methods work on memory blocks, individual files, etc. Streams have two properties, size and position, that denote the stream's size in bytes and the current position within the bytes. Streams also provide methods for reading, writing, and copying bytes into and out of the stream.

It may help to think of a stream as you would magnetic audio tape. Sound is recorded sequentially on tape in much the same way that data is stored in a stream. Continuing this metaphor, a tape counter is similar to the stream position property because it indicates the current position on the tape. To hear the sounds, or obtain data from the stream, you must ensure you "rewind" the position property back to where you want to start, usually position 0 for the beginning.

BLOb Streams to the Rescue

The key to manipulating memo fields in Object Pascal is the *TBlobStream* class. BLOb streams provide an easy way to access or modify a memo field by allowing you to read or write to the field as if it were a file or stream.

The *Create* constructor is used to link the field to the BLOb stream. Here is its syntax:

```
constructor Create(Field : TBlobField;
                   Mode : TBlobStreamMode );
```

Create has two parameters. The first is a reference that points to the memo field to be manipulated. The second specifies the read-write *mode* with one of the *TBlobStreamMode* constants: *bmRead*, *bmWrite*, or *bmReadWrite*. As Delphi's online documentation states: use *bmRead* to access an existing memo field, *bmWrite* to clear the contents of the field and assign a new value, and *bmReadWrite* to modify an existing value. After the BLOb stream is created, the data

can be accessed. To do this, we use the stream methods available to all classes derived from the *TPersistent* class. Note that many of these objects contain stream calls that are not documented in Delphi's online help.

MemoToList: Reading Memo Fields into Memory

First, we'll discuss how to read data from a database into memory where it can be readily used. Then we'll go over the steps needed to write data to a memo field in the database.

To read data into memory, we use the *LoadFromStream* procedure. Here is its syntax:

procedure LoadFromStream(*Stream*: TStream);

This will load data from the specified stream into the object. The *MemoToList* procedure creates a *BlobStream* in read mode and then pulls the contents of a memo field into a *TStringList*:

```
procedure MemoToList(SourceTable:  TTable;
                     SourceField:  string;
                     DestList:     TStringList);
var
  BlobStream1: TBlobStream;
begin
  BlobStream := TBlobStream.Create(TMemoField(
    SourceTable.FieldByName(SourceField)),bmRead);
  DestList.LoadFromStream(BlobStream1);
end;
```

As you can see, *MemoToList* takes three parameters. The first is a *TTable* that should be attached to the table and positioned on the record you want to read. The second is a *String* containing the field name from the table to read, and the third is the *TStringList* that will contain the text read back from the memo field. In the *Create* constructor, we use the *TMemoField* to inform the compiler that *SourceField* is actually a memo field.

ListToMemo: Writing Memo Fields to a Table

Now that we can read data from the memo field, we must be able to put data into it. As in the previous example, we have to create a *TBlobStream* that references the field we want to manipulate. This time, we'll use *bmWrite* as the mode. This will clear the field before the data is inserted. We then save the text from the string list to the BLOb stream. The syntax of the *Write* method is:

function Write(const *Buffer* ; *Count* : Longint) : Longint;

Write requires two parameters: *Buffer*, which is a block of memory that contains the string you want to write, and *Count*, which is the number of characters (bytes) to be written. *Write* is a function and returns the number of characters that were written.

Figure 1 shows the source code for the *ListToMemo* procedure. In our example, the text we want to write to the memo is contained in a list. We can get a pointer to the memory that contains the text by using the *GetText* function. This will return the address in memory where the string starts. Because the *Write* function requires the actual text as the *Buffer* parameter, we attach a caret character (^) to the *GetText* call. This instructs Delphi to use the actual string that *GetText* is pointing to.

```
procedure ListToMemo(DestTable:  TTable;
                     DestField:  string;
                     SourceList: TStringList);
var
  BlobStream1: TBlobStream;
begin
  BlobStream1 :=
    TBlobStream.Create(TMemoField(
      DestTable.FieldByName(DestField)),bmWrite);

  try
    BlobStream1.Write(SourceList.GetText^,
                      StrLen(SourceList.GetText));
  finally
    BlobStream1.Free;
  end;

end;
```

Figure 1: The *ListToMemo* procedure.

We also need to pass the size of the string. We do this by using the *StrLen* function. *StrLen* takes a null-terminated string as a parameter and returns the number of characters contained in the string. So, we pass the text from the *TStringList* we retrieve to the *StrLen* function by using the *GetText* function again. The call to *Write* resembles this:

```
BlobStream1.Write(SourceList.GetText^,
  StrLen(SourceList.GetText));
```

That's it. You now have the code to read and write text to and from memo fields in a database. For convenience, I placed the *MemoToList* and *ListToMemo* procedures in a unit called *MemoList*. When I need their functionality in a Delphi form, I include the *MemoList* unit in the form's **uses** clause.

This article is accompanied by the sample Memo I/O application (see Figure 2) that uses these two routines. The form contains two *TMemo* components (*Memo1* and *Memo2*), two *TButtons* (one to save and the other to load a button), and a *TTable*.

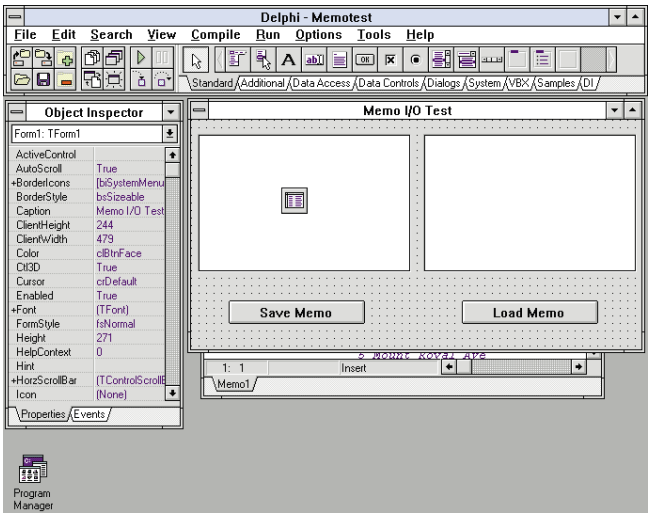


Figure 2: The sample Memo I/O application.

The **Save Memo** button writes the contents of *Memo1* to a memo field in a sample Paradox table using the *ListToMemo* function. Memo controls have a *Lines* property that is a kind of string list, allowing me to cast the *Lines* property to a *TStringList* so I can properly call the *ListToMemo* method. The *OnClick* method for the **Save Memo** button is similar to this:

```
procedure TForm1.SaveButtonClick(Sender: TObject);
begin
  Table1.Open;
  Table1.Edit;
  ListToMemo(Table1,'MemoField',TStringList(Memo1.Lines));
  Table1.Post;
end;
```

Also, notice we need to put the table into edit state before calling the *ListToMemo* function. We can do this by using the table's *Edit*, *Append*, or *Insert* method before calling *ListToMemo*. Finally, we must post the changed record after the *ListToMemo* call. The call to *MemoToList*, the **Load Memo** button's *Click* method, looks similar to the *ListToMemo* call above. Remember the *TTable* must be on the appropriate record that you want to read.

Conclusion

We have discussed getting data into and out of tables. However, we haven't addressed the types of tables that can be used. The sample Memo I/O program uses a Paradox table to store the data. The *ListToMemo* and *MemoToList* methods can also be used on SQL server tables. I have successfully used these routines on InterBase, Microsoft SQL Server, and Sybase tables. There is no reason it shouldn't work on other database server types that are available through the Borland Database Engine.

At first glance, streams can be a complex and confusing issue. To make matters worse, there's little to no documentation on this extremely useful type of object. Once you understand their functionality however, you will likely use them all the time. By using the methods described above, you can add memo fields to all your databases and not have to worry about how to fill them.

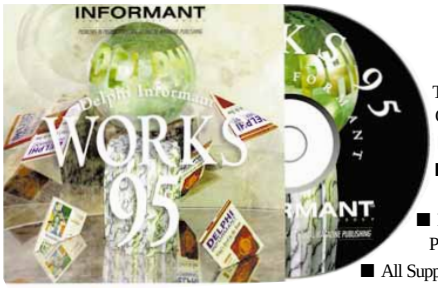
ListToMemo and *MemoToList* are extremely handy routines that I use on most of my development projects. The same techniques can be used to store and retrieve graphics, data points, and even other objects — but that's for another article. Δ

The Memotest project is available on the Delphi Informant Works CD located in INFORM\96\APR\DI9604DK.

Dana Kaufman is a Senior Consultant with Apogee Information Systems, Inc., a Massachusetts-based consulting and development firm specializing exclusively in Delphi and Paradox applications. He is a contributing technical editor for QUE on their Delphi 2 products. Dana can be reached at (508) 481-1400 or via the Internet at aisys@ix.netcom.com.

The Must Have Reference Source For The Serious Delphi™ Developer

The Entire Text of Over 60 Technical Articles Appearing in Delphi™ Informant® in 1995



The Delphi™ Informant® Works 1995 CD-ROM Includes:

- Over 60 Technical Articles
- Text and Keyword Search Capability
- Improved Speed and Performance
- All Supporting Code and Sample Files
- 16-Page Color Booklet
- Electronic Version of *Delphi Power Tools* Catalog
- Electronic Version of *Borland C++ Power Tools* Catalog
- Third-Party Add-In Product Demos
- CompuServe Starter Kit with $15 Usage Credit.

A $105 Value Available Now for only

$39.95

California residents add 7¼% Sales Tax, plus $5 shipping & handling for US orders. (International orders add $15 shipping & handling)

Call Now Toll Free 1-800-88-INFORM

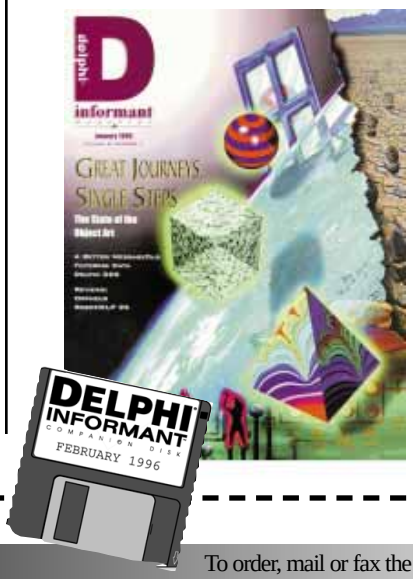
1-800-884-6367 Ask for offer # PI95 To order by mail, send check or Money Order to:
Informant Communications Group, Inc.
ATTN: Works CD offer # PI95
10519 E. Stockton Blvd, Suite 142 Elk Grove, CA 95624-9704
or Fax your order to
916-686-8497

Get Informed!

Subscribe to Delphi Informant, The Complete Monthly Guide to Delphi Development and stay ahead of the rapid application development curve.

Order Now and Get One Issue FREE!

For a limited time you can receive the first issue FREE plus 12 additional issues for only $49.95 That's nearly 25% off the yearly cover price!



Each big issue of *Delphi Informant* is packed with Delphi tips, techniques , news, and more!

- Client/Server Application Development
- Delphi Programming
- Using the Borland Database Engine
- Object -Oriented Programming
- Creating Reusable Components
- Product Reviews
- News from the Delphi Community
- Delphi User Group Information

Payment Method...

- ❑ Check (payable to Informant Communications Group)
- ❑ Purchase Order-- Provide Number _____
- ❑ Visa
- ❑ Mastercard
- ❑ American Express

_____
Card Number

_____
Expiration Date

_____
Signature

International rates

Magazine-only
$54.95/year to Canada
$74.95/year to Mexico
$79.95/year to all other countries

Magazine AND Companion Disk Subscriptions
$124.95/year to Canada
$154.95/year to Mexico
$179.95 to all other countries

California Residents add 7¼% sales tax on disk subscription

☐ YES

To order, mail or fax the form below or call (916) 686-6610 Fax: (916) 686-8497

I want to sharpen my Delphi Programming skills. I've checked the subscription plan I'm interested in below

☐ **Magazine-Only Subscription Plan...**
13 Issues Including One Bonus Issue at $49.95.

☐ **Magazine AND Companion Disk Subscription Plan...**
13 Issues and Disks Including One Bonus Issue and Disk at $119.95
The Delphi Informant Companion Disk contains source code, support files, examples, utilities, samples, and more!

☐ **Delphi Informant Works 1995 CD-ROM $39.95**
US residents add $5 shipping and handling. International customers add $15 shipping and handling.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

Country _____ Phone _____

FAX _____ e-Mail _____

PI95